Serangan Kolisi dan *Second Preimage* Terhadap Skema Modifikasi MDC-2 Berbasis PRESENT

Anjeli Lutfiani[1)], Bety Hayat Susanti[2)]
(1) Badan Siber dan Sandi Negara, anjeli.lutfiani@bsssn.go.id
(2) Program Studi Rekayasa Kriptografi, Politeknik Siber dan Sandi Negara, bety.hayat@poltekssn.ac.id

Abstrak

MDC-2 merupakan salah satu bentuk skema double-length hash function yang bertujuan untuk memfasilitasi jaminan integritas data. Skema ini menggunakan input berupa dua operasi block cipher per blok dari input hash dengan hanya menggunakan input berupa pesan tanpa kunci. Ide konstruksi penggunaan block cipher sebagai komponen utama fungsi hash diharapkan dapat menyediakan efisiensi yang sama jika implementasi suatu block cipher dikatakan efisien. Tiga sifat utama agar fungsi hash dapat dikatakan aman adalah resistensi preimage, resistensi preimage kedua, dan resistensi kolisi. Pada penelitian ini, dilakukan desain modifikasi konstruksi MDC-2 menggunakan basis fungsi kompresi berupa algoritme lightweight block cipher PRESENT yaitu DOPE (Double-length Matyas-Meyer-Oseas based on PRESENT). DOPE menggunakan skema Matyas-Meyer-Oseas dengan tambahan komponen berupa padding, fungsi permutasi dan fungsi transformasi linier. Analisis dilakukan terhadap sifat resistensi preimage atau one-wayness melalui pembuktian, resistensi preimage kedua melalui second preimage attack, dan resistensi kolisi melalui fixed point attack. Berdasarkan pembuktian menggunakan upaya reverse terhadap skema, DOPE dikatakan dapat memenuhi sifat one-wayness. Implementasi kedua serangan terhadap dua variasi penggunaan fungsi utama miniatur DOPE menggunakan PRESENT berukuran 32-bit dan 64-bit menunjukkan bahwa tidak ditemukan adanya kolisi. Tiga kolisi ditemukan pada second preimage attack terhadap variasi DOPE berbasis algoritme PRESENT berukuran 16-bit. Probabilitas ditemukannya kolisi pada variasi tersebut adalah sebesar $6.984919309616089\times10^{-10}$. Kolisi pada DOPE terbentuk setelah proses XOR antara pesan awal dengan output pada tahap fungsi enkripsi.

Kata kunci: *fixed point attack, MDC-2, one-wayness, PRESENT, second preimage attack.*

Abstract

MDC-2 is a form of double-length hash function scheme that aims to facilitate data integrity guarantees. This scheme uses input in the form of two block cipher operations per block of hash input, using only input in the form of messages without keys. The construction idea of using a block cipher as the main component of the hash function is expected to provide the same efficiency as if the implementation of a block cipher were said to be efficient. The three main properties for which a hash function can be said to be safe are preimage resistance, second preimage resistance, and collision resistance. In this research, a modification design was carried out for the MDC-2 construction using a compression function base in the form of the lightweight block cipher PRESENT algorithm, namely DOPE (Double-length Matyas-Meyer-Oseas based on PRESENT). DOPE uses the Matyas-Meyer-Oseas scheme with additional components in the form of padding, permutation functions, and linear transformation functions. Analysis was carried out on the nature of preimage resistance or one-wayness through proof, second preimage resistance through second preimage attacks, and collision resistance through fixed point attacks. Based on evidence using reverse efforts on the scheme, DOPE is said to be able to fulfill one-way characteristics. The implementation of both attacks on two variations of the main DOPE miniature function using 32-bit and 64-bit PRESENT showed that no collision was found. Three collisions were found in the second preimage attack against DOPE variations based on the 16-bit PRESENT algorithm. The probability of finding a collision in this variation is $6.984919309616089\times10^{-10}$. A collision in DOPE is formed after the XOR process between the initial message and the output at the encryption function step.

Keywords: *fixed point attack, MDC-2, one-wayness, PRESENT, second preimage attack.*

1. PENDAHULUAN

Fungsi *hash* merupakan penyedia layanan integritas data yang memetakan *input* dengan panjang sembarang menjadi *output* dengan panjang tetap [1][2]. Dalam membangun fungsi *hash*, diperlukan fungsi yang dapat memetakan *input* ke suatu himpunan *output* yang terdistribusi acak [3]. Tiga sifat utama agar suatu fungsi *hash* dapat dikatakan aman adalah resistensi *preimage*, resistensi *preimage* kedua, dan resistensi kolisi [4].

Berdasarkan penggunaan kunci, fungsi *hash* terbagi menjadi *unkeyed hash function* dan *keyed hash function*. Subkelas dari *keyed hash function* adalah *Modification Authentication Code* (MAC) sedangkan subkelas dari *unkeyed hash function* adalah *Modification Detection Code* (MDC) [1]. MDC merupakan fungsi *hash* yang bertujuan untuk memfasilitasi jaminan integritas data dengan hanya menggunakan *input* berupa pesan tanpa kunci [1].

Fungsi *hash* dapat dibangun dengan menggunakan basis utama dari konstruksi berupa *block cipher* sebagai fungsi kompresi [1][5]. Kategori ini terbagi menjadi konstruksi fungsi *hash* yang menghasilkan nilai *hash* berukuran *single-length* (n-bit) yaitu berukuran sama panjang dengan ukuran panjang *block cipher* dan *double-length* ($2n$-bit) yang berukuran dua kali ukuran panjang *block cipher* [1].

Ide konstruksi fungsi *hash* berbasis *block cipher* berawal dari gagasan bahwa jika implementasi suatu *block cipher* dikatakan efisien, maka penggunaan *block cipher* sebagai komponen utama fungsi *hash* diharapkan dapat menyediakan efisiensi yang sama [6]. Nilai *hash* yang dihasilkan dari *input* pesan merupakan *output* dari iterasi terakhir pada fungsi kompresi yang menjadi basis [7].

Skema-skema konstruksi fungsi *hash* berbasis *block cipher* diantaranya adalah skema Davies-Meyer, Miyaguchi-Preneel dan Matyas-Meyer-Oseas [8]. Berdasarkan [1], implementasi penggunaan konstruksi fungsi *hash* berukuran *single-length* dengan menggunakan basis *block cipher* diketahui tidak tahan terhadap kolisi. Selanjutnya untuk mengatasi hal tersebut, dibangun skema MDC-2 berupa metode konstruksi fungsi *hash* dengan nilai *hash* berukuran dua kali ukuran *block cipher*.

Sesuai dengan tujuannya, MDC-2 diharapkan dapat memiliki ketahanan yang baik terhadap kolisi. Hal tersebut berarti bahwa untuk n-bit *block* yang menghasilkan $2n$-bit nilai *hash*, usaha yang diperlukan untuk menemukan kolisi dalam fungsi *hash* diharapkan menjadi 2^n [9]. Namun dalam tesisnya, Thomsen menyatakan bahwa skema umum MDC-2 diketahui rentan terhadap serangan kolisi dan serangan *preimage* [9].

Pada tahun 2008, A. Bogdanov *et al*. melakukan penelitian konstruksi fungsi *hash* berbasis *block cipher* dengan berbagai variasi menggunakan algoritme AES dan PRESENT sebagai komponen utamanya. Pada penelitian tersebut, disimpulkan bahwa H-PRESENT-128 merupakan kandidat terbaik di antara variasi lainnya [10]. Algoritme tersebut juga hanya membutuhkan siklus *clock* sebanyak 20 hingga 30 kali lebih sedikit dibandingkan dengan algoritme AES dan MD5 [10].

Hirose dan Kuwakado melakukan penelitian analisis terhadap konstruksi fungsi *hash* berukuran *double-length* berbasis AES-192 atau AES-256 [11]. Analisis perbandingan yang dilakukan adalah analisis ketahanan terhadap sifat resistensi kolisi, resistensi *preimage*, dan sifat lainnya. Hasil penelitian menunjukkan bahwa skema fungsi *hash* berukuran *double-length* memiliki ketahanan yang baik [11]. Pada penelitian lain, Bos *et al.* menunjukkan bahwa konstruksi fungsi *hash* dengan skema Davies Meyer dan skema *single-length* lainnya dapat memberikan keuntungan dalam eksploitasi fitur perangkat keras [12].

Berdasarkan kekurangan dari skema MDC-2 serta hasil implementasi penggunaan algoritme PRESENT dan skema Matyas-Meyer-Oseas pada beberapa variasi skema fungsi *hash* pada penelitian-penelitian sebelumnya, pada penelitian ini dilakukan modifikasi konstruksi skema MDC-2. Konstruksi modifikasi dilakukan dengan menggunakan skema Matyas-Meyer-Oseas berbasis algoritme PRESENT [13], yang selanjutnya disebut dengan algoritme DOPE (*Double-length Matyas-Meyer-Oseas based on PRESENT*). Pada penelitian awal, telah dilakukan pengujian terhadap DOPE menggunakan *Yuval's Birthday Attack* [14]. *Yuval's Birthday Attack* merupakan salah satu metode serangan terhadap sifat resistensi kolisi dengan memanfaatkan teori *birthday paradox* dalam melakukan pencarian kolisi [1].

Yuval's Birthday Attack dilakukan terhadap 120 kemungkinan pasangan dari *input* satu blok nilai ekstrem dan *input* satu blok nilai *pseudo random* serta 16 pasang *input* dua blok nilai ekstrem pada DOPE. Setiap pasangan *input* mengambil sampel modifikasi minor dengan mengubah 32-bit terakhir dari pesan. Hasil serangan menunjukkan tidak adanya kolisi yang terjadi dari seluruh kemungkinan nilai *hash* yang dihasilkan.

Berdasarkan hasil dari pencarian kolisi tersebut, pada penelitian ini, analisis pengujian dilakukan dengan menggunakan metode *fixed point attack* dan *second preimage attack*. Kedua pengujian tersebut dilakukan sebagai variasi serangan dalam pencarian kolisi pada DOPE dengan bentuk pengujian yang berbeda. Pembuktian sifat *one-wayness* atau resistensi *preimage* dilakukan terlebih dahulu dengan melakukan upaya *reverse* terhadap struktur DOPE. Selanjutnya *fixed point attack* dilakukan sebagai metode serangan terhadap sifat resistensi kolisi [1] sedangkan *second preimage attack* digunakan sebagai metode serangan terhadap sifat resistensi *preimage* kedua [1].

2. LANDASAN TEORI

Bagian ini menjelaskan mengenai skema MDC-2, algoritme *ultra-lightweight block cipher*, dan serangan terhadap fungsi *hash.*

2.1 Fungsi *Hash*

Fungsi *hash* merupakan fungsi yang memetakan *input* bit *string* dengan panjang sembarang menjadi *output* dengan panjang tetap [2][9]. Hasil *output* dari fungsi *hash* disebut sebagai *hash code*, *hash result*, *hash value*, atau hanya disebut sebagai *hash* [1]. Dua syarat minimal yang harus dimiliki suatu fungsi *hash* adalah memenuhi sifat kompresi dan mudah dihitung [3].

Tiga sifat utama agar suatu *unkeyed hash function* dapat dikatakan aman, selain kompresi dan mudah dihitung dijelaskan berdasarkan definisi pada [7]. Resistensi *preimage* atau *one-wayness* menjamin bahwa apabila untuk suatu nilai *hash* H, maka secara komputasi sulit untuk menemukan pesan *input* x yang memenuhi $H = f(x)$, dengan f merupakan suatu fungsi *hash*. Resistensi *preimage* kedua atau *weak-collision resistance* terpenuhi ketika untuk suatu *input* pesan x_1, sulit untuk menemukan *input* pesan x_2 yang memenuhi $f(x_1) = f(x_2)$, dengan x_1 tidak sama dengan x_2. Resistensi kolisi atau *strong collision resistance* merupakan kondisi bahwa secara komputasi sulit untuk menemukan pesan *input* x_1 dan x_2 yang memiliki nilai *hash* sama, dengan $x_1 \neq x_2$.

2.2 Skema MDC – 2

MDC-2 merupakan salah satu bentuk skema *double-length hash function* yang bekerja dengan menggunakan *input* berupa dua operasi *block cipher* per blok dari *input hash*. Penggunaan *block cipher* sebagai basis dari *hash function* dapat meminimalisir usaha dalam melakukan desain dan implementasi fungsi *hash* [6]. Dalam [1], ketiga skema konstruksi fungsi *hash* berukuran *single-length* [8] tersebut terbukti aman pada model pengujian *black box* dengan penggunaan basis *block cipher* yang memenuhi syarat sebagai *random one-way permutation.*

Namun berdasarkan [1], diketahui bahwa penggunaan *block cipher* berukuran 64-bit sebagai basis konstruksi fungsi *hash* menghasilkan nilai *hash* yang tidak tahan terhadap kolisi. Selanjutnya penggunaan *block cipher* tersebut dikonstruksi menjadi fungsi *hash* yang menghasilkan nilai *hash* dengan panjang $2n$-bit atau mendekati ukuran 128-bit [1]. Tujuan konstruksi fungsi *hash* berukuran *double-length* ini diharapkan dapat menghasilkan fungsi *hash* yang memiliki ketahanan yang baik terhadap kolisi [9].

2.3 Algoritme PRESENT dan Hummingbird

Algoritme PRESENT merupakan algoritme yang memetakan 64-bit blok *pesan* menjadi 64-bit blok *ciphertext* [13]. Keseluruhan proses algoritme PRESENT berisi 32 *round* yang terdiri atas pembangkitan kunci atau *Key Schedule*, operasi XOR pada addRoundKey, operasi substitusi non-linier pada sBoxLayer, dan permutasi linier pada pLayer. Algoritme PRESENT menggunakan kunci rahasia berukuran 80-bit yang dioperasikan ke dalam proses pembangkitan kunci untuk menghasilkan 32 subkunci yang digunakan sebanyak satu subkunci pada setiap *round*. Tahapan pada *round* ke-32 hanya berisikan tahap *post-whitening* berupa operasi XOR menggunakan subkunci ke-32 tanpa melalui tahap sBoxLayer dan pLayer.

Algoritme Hummingbird merupakan kombinasi dari *block cipher* dan *stream cipher* dengan blok berukuran 16-bit, kunci berukuran 256-bit dan *internal state* berukuran 80-bit [15]. Struktur keseluruhan algoritme Hummingbird terdiri atas empat *block cipher* berukuran 16-bit dan empat register *internal state* 16-bit serta 16-*stage* LFSR. Kunci berukuran 256-bit dibagi menjadi empat subkunci berukuran 64-bit yang digunakan pada keempat *block cipher*. Sebelum melakukan proses enkripsi, dilakukan terlebih dahulu proses inisialisasi terhadap empat *nonce* acak berukuran 16-bit untuk menghasilkan register *internal state* dan *input* LFSR.

2.4 *Fixed Point Attack*

Materi mengenai *fixed point attack* ini merujuk pada [1] kecuali disebutkan lain. Penggunaan *block cipher* sebagai basis utama konstruksi fungsi *hash* menjadikan fungsi *hash* yang digunakan mendapat perhatian lebih pada penggunaannya. Bahaya umum yang mungkin terjadi adalah bahwa sifat pada suatu *block cipher* dapat memfasilitasi manipulasi *input* fungsi kompresi seperti pada sifat *fixed point*. Dalam penelitiannya, Hirose menyebutkan bahwa dalam melakukan konstruksi fungsi *hash*, basis *block cipher* atau fungsi kompresi lain yang mendasari perlu untuk diperhatikan [16].

Fixed point pada *block cipher* merupakan sifat yang memungkinkan penyerang untuk melakukan *fixed point attack* jika penyerang dapat mengontrol *input* kunci dari *block cipher* yang digunakan. Dalam penggunaan *block cipher* pada fungsi kompresi, sifat ini dapat digunakan untuk menemukan kolisi dalam fungsi *hash* [17]. Sifat *fixed point* pada fungsi kompresi $f(H_{i-1}, x_i) = H_i$ merupakan pasangan (H_{i-1}, x_i) sedemikian hingga $H_{i-1} = f(H_{i-1}, x_i)$ dengan H_{i-1} merupakan *initial value* dan x_i merupakan pesan ke-i [18].

2.5 *Second Preimage Attack*

Resistensi *preimage* kedua merupakan salah satu sifat yang harus terpenuhi agar suatu fungsi *hash* dikatakan aman [4]. Hal ini berarti untuk n-bit nilai *hash*, usaha yang diperlukan untuk menahan serangan *second preimage* adalah sekitar 2^n. Dalam melakukan serangan terhadap suatu pesan, penyerang harus melakukan usaha sekitar 2^n untuk menemukan pesan lain yang memiliki nilai *hash* yang sama dengan nilai yang dihasilkan.

Second preimage attack merupakan serangan dasar terhadap sifat resistensi *preimage* kedua [1]. Apabila penyerang ingin menemukan pesan lain yang memiliki nilai *hash* yang sama dengan nilai *hash* dari pesan sebenarnya, maka penyerang harus menemukan suatu pesan sedemikian hingga $H_i = f(H_{i-1}, x'_i)$ menghasilkan nilai *hash* dari x' yang sama dengan nilai *hash* dari *chosen message* yang dimiliki [19].

3. METODE PENELITIAN

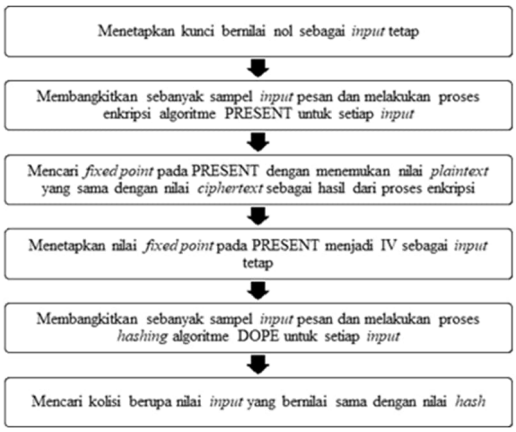
Metode penelitian yang digunakan adalah metode kuantitatif dengan melakukan kajian kepustakaan dan eksperimen berupa perancangan dan simulasi skema modifikasi, vektor uji, pembuktian dan pengujian ketahanan algoritme DOPE (*Double-length* Matyas-Meyer-Oseas *based on* PRESENT) pada tiga sifat utama fungsi *hash*. Pada penelitian ini proses pengolahan data dilakukan dengan menggunakan bahasa pemrograman Python 3.10. Simulasi pada penelitian ini menggunakan sistem operasi Windows 10 Pro dengan *processor* Intel Core i7-10700 dan *Random Acess Memory* (RAM) sebesar 16,0 GB dengan pemrograman Python 3.10 berbasis paralel guna mempercepat proses komputasi serangan.

Berdasarkan metode penelitian yang disebutkan sebelumnya, tahapan pada penelitian ini dijelaskan sebagai berikut:

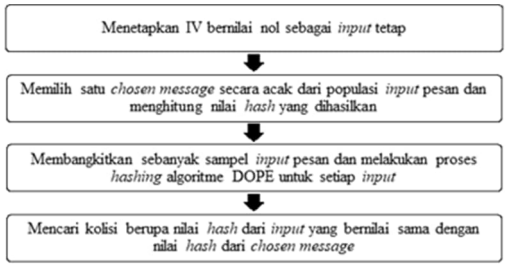
a. Penelitian diawali dengan kajian kepustakaan yang dilakukan dengan mencari, mengumpulkan, dan mempelajari berbagai referensi terkait konsep

dasar dan teori-teori yang berkaitan dengan penelitian ini.

b. Perancangan DOPE diawali dengan melakukan penambahan tahap praproses berupa proses *padding*, penyesuaian tahap fungsi permutasi pada DOPE dan dilakukan perubahan fungsi enkripsi menjadi algoritme PRESENT. Selain itu, pada tahap akhir dilakukan penambahan fungsi transformasi linier dari algoritme Hummingbird.
c. Pembuktian sifat *one-wayness* atau resistensi *preimage* dibuktikan dengan menunjukkan bahwa tidak dapat dilakukan proses pembalikan (*reverse*) terhadap algoritme DOPE.
d. Pada penelitian ini, implementasi *fixed point attack* dan *second preimage attack* diterapkan ke dalam tiga variasi penggunaan fungsi utama yang berbeda yaitu variasi algoritme PRESENT berukuran 16-bit, 32-bit, dan 64-bit. Pengujian keamanan dilakukan terhadap miniatur algoritme DOPE berupa variasi pesan berukuran 32-bit, 64-bit dan 128-bit yang disebut sebagai DOPE – [4], DOPE – [8], dan DOPE – [16] secara berurutan. Skenario serangan ditunjukkan pada Gambar 1 dan Gambar 2.



Gambar 1. Skenario Tahapan *Fixed Point Attack*



Gambar 2. Skenario Tahapan *Second Preimage Attack*

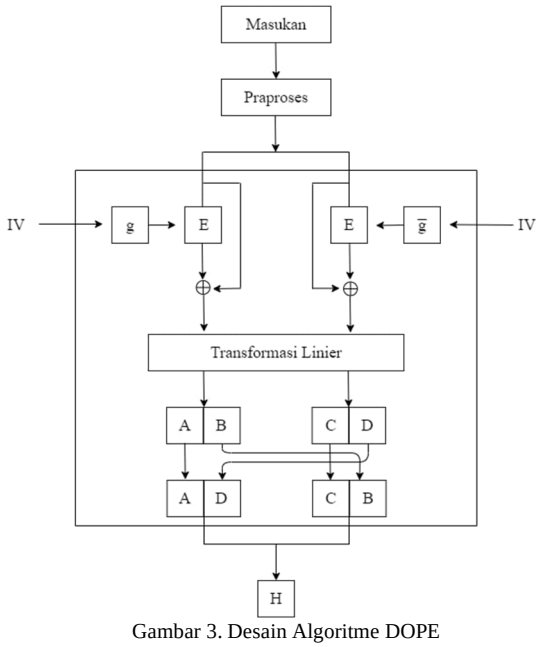
e. Setelah melakukan perancangan skenario serangan, selanjutnya dilakukan penerapan metode pengujian ketahanan terhadap kolisi berupa *fixed point attack* dan pengujian ketahanan terhadap sifat *second preimage* berupa *second preimage attack.*
f. Analisis hasil pengujian ketahanan DOPE terhadap kolisi berupa *fixed point attack* dan ketahanan terhadap *second preimage* berupa *second preimage attack* dilakukan dengan menghitung probabilitas ditemukannya kolisi pada kedua serangan dari besar usaha yang dilakukan.
g. Pada tahap terakhir penelitian, dilakukan pengambilan simpulan mengenai hasil konstruksi DOPE, pemenuhan sifat resistensi *preimage* serta ketahanan DOPE atas serangan terhadap sifat kolisi dan *second preimage.*

4. HASIL DAN PEMBAHASAN

Bagian ini menjelaskan hasil penelitian yang dibahas berisi pembahasan desain skema modifikasi, pembuktian skema *one-wayness* serta pembahasan mengenai hasil dan analisis terhadap implementasi serangan pada algoritme DOPE.

4.1 Desain Algoritme

Algoritme DOPE (*Double-length Matyas-Meyer-Oseas based on PRESENT*) sebagaimana terlihat pada Gambar 3 memproses blok pesan berukuran 128-bit atau kelipatannya dengan *input* IV yang digunakan sebagai dua kunci berukuran 80-bit pada fungsi utamanya setelah melalui tahapan fungsi permutasi g dan $\bar{g}$. Selanjutnya pasangan blok *pesan* dan kunci masing-masing diproses pada algoritme PRESENT sebagai fungsi enkripsi yang dinotasikan sebagai E. Setelah tahap transformasi linier, dilakukan operasi *swap* terhadap *nibble* blok A, B, C, dan D yang dihasilkan seperti pada Gambar 3. Keluaran dari DOPE merupakan hasil *swap* berupa nilai *hash* berukuran panjang tetap 128-bit.



Gambar 3. Desain Algoritme DOPE

Penggunaan skema konstruksi MDC-2 didasarkan dari banyak munculnya perangkat dengan ukuran lingkungan terbatas seperti RFID atau *smartcard.* Pada dasarnya, seorang perancang *hardware* harus mengimplementasikan hanya satu *block cipher* untuk menghasilkan sebuah fungsi enkripsi yang setara dengan fungsi *hash* [20]. Namun,

karena ukuran panjang fungsi enkripsi dianggap terlalu kecil untuk memenuhi sifat ketahanan terhadap kolisi, maka dibangun suatu fungsi *hash* yang dapat menghasilkan nilai *hash* berukuran panjang ganda ($2n$-bit) atau *double-length hash function*.Algoritme DOPE diawali dengan tahap praproses yang berguna untuk menyesuaikan ukuran *input* IV menjadi berukuran 64-bit atau kelipatannya dan ukuran *input* pesan menjadi berukuran 128-bit atau kelipatannya. Proses *padding* yang digunakan pada DOPE merupakan metode *zero byte padding* yang dispesifikasikan penggunaannya pada fungsi *hash* dalam ISO/IEC 10118-1 [1]. Pada tahap ini, *input* pesan dan IV yang memiliki panjang acak dan tidak berukuran sesuai selanjutnya dilakukan *padding* 0* atau penambahan biner nol hingga ukuran panjang terpenuhi. Selanjutnya DOPE juga membagi *input* pesan dan IV ke dalam dua bagian berbeda untuk diproses seperti pada Gambar 3. Apabila *input* IV hanya terdiri atas satu blok 64-bit, maka IV selanjutnya digunakan pada kedua bagian algoritme dan diproses pada tahap fungsi permutasi secara terpisah.

Fungsi permutasi pada algoritme DOPE bertujuan untuk menyesuaikan ukuran IV dengan ukuran kunci yang diperlukan untuk digunakan pada tahapan selanjutnya serta memperoleh difusi. Proses pembangkitan kunci algoritme PRESENT pada tahapan fungsi enkripsi E memerlukan *input* kunci berukuran 80-bit untuk menghasilkan sejumlah 32 subkunci yang digunakan dalam setiap *round* pada fungsi enkripsi. Fungsi g dan $\bar{g}$ pun dibuat berbeda antara blok kiri dengan blok kanan guna memastikan apabila IV yang dijadikan *input* memiliki nilai yang sama, maka hasil permutasi yang selanjutnya menjadi kunci untuk tahap fungsi enkripsi E tidak kembali menghasilkan nilai yang sama. Fungsi permutasi g dan $\bar{g}$ bekerja sebagai berikut:

$$g(IV) : IV_A \mid IV_B \mid 00000000\ 00000001 \mid IV_C \mid IV_D$$

$$\bar{g}(IV) : IV_A \mid IV_B \mid 10000000\ 00000000 \mid IV_C \mid IV_D$$

Penggunaan algoritme PRESENT sebagai fungsi utama dari DOPE didasarkan pada fakta bahwa PRESENT merupakan algoritme *lightweight* berbasis SPN yang memiliki *input* dan kunci dengan ukuran yang berbeda. Sesuai dengan tujuan konstruksi penggunaan skema MDC-2, algoritme PRESENT telah terbukti baik untuk diimplementasikan pada perangkat dengan ukuran lingkungan terbatas seperti perangkat RFID dan *smartcard* [8]. Perangkat-perangkat tersebut biasanya digunakan untuk perlindungan akses pada area terbatas seperti ruangan atau gedung. Skema SPN yang menjadi dasar algoritme PRESENT juga dapat melakukan komputasi lebih cepat dibanding penggunaan skema Feistel seperti pada DES.

Berbeda dengan MDC-2, DOPE tidak hanya melakukan permutasi namun juga melakukan transformasi linier berupa transformasi dari hasil operasi XOR antara hasil enkripsi dengan *input* awal proses enkripsi menjadi *output* berupa empat *nibble* berukuran 16-bit. Transformasi linier berupa rotasi dan operasi XOR memiliki tujuan yang sama yaitu untuk memperoleh difusi. Pada DOPE, pemilihan fungsi transformasi linier didasarkan terhadap penggunaan fungsi transformasi linier pada algoritme Hummingbird. Pemilihan tersebut dilakukan karena keduanya merupakan algoritme *lightweight*. Nilai rotasi yang dipilih adalah 5 dan 11, sehingga rotasi tidak menyebabkan bit kembali ke posisi awal.

Setelah dilakukan operasi XOR antara hasil enkripsi dengan *input* awal proses enkripsi, hasil operasi tersebut dibagi menjadi empat *nibble* berukuran 16-bit untuk tahap transformasi linier. Pada tahapan ini dilakukan operasi XOR antara hasil dari operasi sebelumnya dengan *nibble* yang telah dilakukan *circular left shift* sebanyak 5-bit dan 11-bit sebagai berikut:

$$L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$$

$$L(c) : c \oplus (c <<< 5) \oplus (c <<< 11)$$

Setelah didapatkan hasil dari tahap transformasi linier, selanjutnya dilakukan operasi *swap* terhadap *nibble* yang dihasilkan seperti pada Gambar 3. Hasil *swap* tersebut selanjutnya disebut sebagai nilai *hash* yang dihasilkan dari algoritme DOPE.

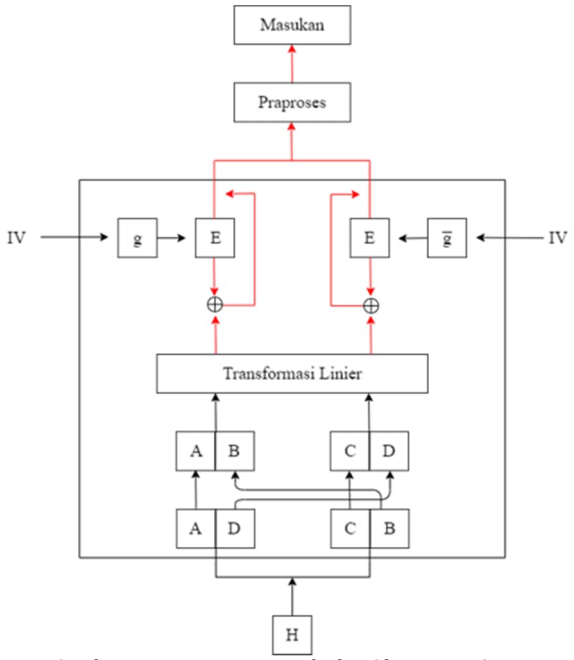
4.2 Pembuktian *One-Wayness*

DOPE merupakan algoritme fungsi *hash* dengan dua operasi blok yang terdiri atas tahap praproses, fungsi permutasi, fungsi enkripsi dan transformasi linier. Pada upaya pencarian suatu nilai *input* dari pengetahuan terhadap suatu nilai *hash*, dapat dilakukan proses *reverse* terhadap struktur skema fungsi *hash*. Alur upaya *reverse* terhadap algoritme DOPE divisualisasikan pada Gambar 4.

Proses *swap* berupa pertukaran posisi blok-blok nilai *hash* hanya memiliki satu kemungkinan nilai prediksi *input* yang benar. Garis hitam menunjukkan bahwa proses tersebut dapat dilakukan secara terbalik dengan hanya satu kemungkinan *input* dan *output*. Pada tahap transformasi linier, upaya *reverse* sulit untuk dilakukan karena per satu bit pada nilai *hash* dapat memiliki empat kemungkinan nilai prediksi *input* yang benar. Hal tersebut terjadi karena tahap transformasi linier terdiri atas operasi XOR terhadap tiga nilai hasil rotasi *input*. Misalnya untuk satu bit *hash* bernilai 1_2, terdapat empat kemungkinan kombinasi yang mungkin yaitu $0 \oplus 0 \oplus 1, 0 \oplus 1 \oplus 0, 1 \oplus 0 \oplus 0$, dan $1 \oplus 1 \oplus 1$. Begitu pula terdapat empat kemungkinan kombinasi untuk satu bit *hash* bernilai 0_2, yaitu hasil operasi nilai $1 \oplus 1 \oplus 0, 1 \oplus 0 \oplus 1, 0 \oplus 0 \oplus 1$, dan $0 \oplus 0 \oplus 0$.

Sehingga untuk satu *nibble* berukuran 16-bit yang diketahui, perlu dilakukan percobaan terhadap keempat kemungkinan nilai dari setiap bit *hash*, lalu nilai tersebut harus menjadi pasangan yang tepat agar

dapat dilakukan operasi XOR. Selanjutnya hal tersebut baru dapat berhasil dilakukan jika pasangan yang dipilih benar merupakan pasangan-pasangan hasil nilai rotasi dari *input* yang benar.



Gambar 4. Proses *Reverse* Terhadap Algoritme DOPE

Selanjutnya pada tahapan XOR *input* awal dengan hasil fungsi enkripsi, nilai *hash* memiliki lebih dari satu kemungkinan nilai prediksi *input* yang benar. Seperti ditunjukkan pada garis merah, hal tersebut terjadi karena hasil dari tahapan fungsi enkripsi yang selanjutnya digunakan pada tahapan XOR tersebut memiliki kemungkinan kombinasi nilai *input* sepanjang 2^n dengan n merupakan ukuran panjang pesan. Meskipun tahapan tersebut hanya berisi operasi XOR, namun perlu upaya yang besar untuk menentukan nilai hasil enkripsi untuk mendapatkan *input* awal. Dua blok yang digunakan pada DOPE juga memperbesar kemungkinan ditemukannya nilai *input* awal secara keseluruhan menjadi sebanyak dua kali lipat.

Secara umum jumlah kemungkinan dalam usaha pencarian nilai *input* apabila diketahui suatu nilai *hash* untuk algoritme DOPE berbasis *block cipher* berukuran n-bit memenuhi persamaan:

$$\text{Jumlah kemungkinan} = 2 \times 2^n + 4n$$

Asumsikan DOPE merupakan skema fungsi *hash* berbasis *block cipher* berukuran 2-bit, sehingga *input* dan *output* dari skema tersebut berukuran 4-bit. Misalnya dari 15 kemungkinan *output hash* yang dihasilkan, diambil suatu nilai *hash* 1100_2 atau C dalam ASCII untuk dilakukan pencarian terhadap *input* yang memungkinkan. Pencarian nilai tersebut dapat dilakukan dengan proses *reverse* terhadap struktur algoritme DOPE yang diketahui. Untuk nilai *hash* 1100_2, dimisalkan setiap bit dianggap sebagai $A||B||C||D$ maka dihasilkan nilai prediksi *input* bernilai 1001_2. Pada proses transformasi linier, terdapat sebanyak 16 kemungkinan nilai prediksi *input* yang benar karena untuk setiap bit dari total 4-bit pada nilai yang diketahui memiliki empat kemungkinan nilai yang benar. Selanjutnya, pada hasil XOR antara *input* asli dan hasil enkripsi terdapat empat pasang *input* yang memungkinkan untuk menghasilkan masing-masing blok kanan dan kiri pada skema. Misalnya untuk nilai 10_2, terdapat kemungkinan pasangan 10_2 dengan 00_2, 00_2 dengan 10_2, 01_2 dengan 11_2 dan 11_2 dengan 01_2. Nilai 01_2 juga terdapat empat kemungkinan pasangan yang benar yaitu, 00_2 dengan 01_2, 10_2 dengan 11_2, 01_2 dengan 00_2, dan 11_2 dengan 10_2. Salah satu kemungkinan *hash* berukuran 4-bit yang diketahui, setidaknya terdapat 24 kemungkinan untuk menemukan hasil *input* yang bersesuaian dengan nilai *hash*.

Hal tersebut menunjukkan bahwa upaya *reverse* untuk mengetahui nilai *input* pesan jika diketahui suatu nilai *hash* merupakan usaha yang sulit. Usaha tersebut akan semakin sulit jika digunakan basis *block cipher* dengan n berukuran besar. Misalnya untuk *block cipher* dengan ukuran $n = 256$, maka total kemungkinan dalam upaya *reverse* adalah sebesar $2 \times 2^{256} + 4(256) = 2^{257} + 2^{10}$ yang secara komputasi akan sangat berat untuk dilakukan.

4.3 Hasil dan Analisis Serangan

Pada bagian ini disajikan hasil dan analisis terhadap implementasi serangan pada algoritme DOPE dengan penggunaan populasi dan sampel seperti pada Tabel 1.

Tabel 1.Populasi dan Sampel Pengujian

Uji	Variasi	Sampel	Populasi
Fixed Point Attack	PRESENT-16	2^{16}	2^{16}
	PRESENT-32	2^{32}	2^{32}
	PRESENT-64	2^{32}	2^{64}
	DOPE – [4]	2^{16}	2^{32}
	DOPE – [8]	2^{32}	2^{64}
	DOPE – [16]	2^{32}	2^{128}
Second Preimage Attack	DOPE – [4]	2^{32}	2^{32}
	DOPE – [8]	2^{32}	2^{64}
	DOPE – [16]	2^{32}	2^{128}

Pada implementasi *second preimage attack*, untuk setiap variasi dipilih *chosen message* bernilai $754b8fa9_{16}$, $f8eb8314dd046f67_{16}$, dan $cb8b21b135249576767f65f9ec9694f7_{16}$ dengan hasil pengujian terlihat pada Tabel 3. Tabel 2 menunjukkan hasil pengujian pada dua iterasi DOPE – [4] yang dilakukan guna menentukan ambang batas tidak ditemukannya kolisi pada DOPE – [4].

Skenario *fixed point attack* dilakukan dengan memanfaatkan properti *fixed point* pada algoritme PRESENT sebagai fungsi enkripsi utama setiap variasi algoritme DOPE. Pencarian properti tersebut dilakukan dengan menggunakan variabel tetap berupa kunci bernilai nol dengan ukuran sampel pengujian seperti pada Tabel 4. Sejumlah properti *fixed point*

yang ditemukan tersebut selanjutnya digunakan sebagai *input* IV pada uji *fixed point attack* terhadap algoritme DOPE dengan hasil uji seperti terlihat pada Tabel 5.

Tabel 2. Hasil Pengujian *Second Preimage Attack* pada Dua Iterasi Algoritme DOPE – [4]

Ukuran sampel	M	M'	Kolisi	Jumlah Kolisi
2^{16}	$0000754b$ $754b0000_{16}$	00000001 00010000_{16}	-	0
		00000002 00020000_{16}	-	
		…	-	
		$0000ffff$ $ffff0000_{16}$	-	
2^{16}	$00008fa98$ $fa90000_{16}$	00000001 00010000_{16}	-	0
		00000002 00020000_{16}	-	
		…	-	
		$0000ffff$ $ffff0000_{16}$	-	
2^{31}	$754b8fa97$ $54b8fa9_{16}$	00000000 00000000_{16}	-	0
		00000000 00000001_{16}	-	
		…	-	
		$ffffffff$ $ffffffff_{16}$	-	

Pada [21], pengujian *fixed point attack* terhadap skema LightMAC berbasis PRESENT diketahui menghasilkan sebanyak enam kolisi dengan menggunakan kunci sebagai variabel kontrol. Skema ini menggunakan algoritme PRESENT dengan kunci berukuran 80-bit dan sejumlah 32 *round* sebagai basis fungsi enkripsi. Berdasarkan hasil penelitian, disimpulkan bahwa kolisi belum terbentuk saat proses enkripsi, namun terjadi pada proses di dalam skema LightMAC. Pesan *fixed point* tersebut terbentuk setelah proses *key whitening* di tahapan terakhir, yaitu proses XOR antara kunci dengan *output* pada *round* terakhir.

Pada penelitian ini, pengujian *fixed point attack* terhadap skema DOPE – [4] dengan penggunaan basis fungsi enkripsi dan variabel kontrol yang sama diketahui tidak menghasilkan adanya pesan *fixed point* yang terbentuk. Namun, pada pengujian *second preimage attack* ditemukan adanya tiga *input* dengan nilai *hash* yang berkolisi dengan *hash* dari *chosen message.*

Peluang ditemukannya kolisi pada hasil *second preimage attack* terhadap variasi DOPE – [4] dapat dihitung dengan membagi jumlah kolisi dengan jumlah sampel yang digunakan. Sejumlah tiga *input* dengan nilai *hash* yang berkolisi dibagi dengan jumlah sampel berukuran 2^{32} mencapai nilai probabilitas sebesar $6.984919309616089 \times 10^{-10}$.

Tabel 3. Hasil Pengujian *Second Preimage Attack* pada Algoritme DOPE

IV	DOPE – [4]					DOPE – [8]					DOPE – [16]				
	Ukuran Sampel Pengujian	M	M'	Kolisi	Jumlah	Ukuran Sampel Pengujian	M	M'	Kolisi	Jumlah	Ukuran Sampel Pengujian	M	M'	Kolisi	Jumlah
0000 0000 0000 0000_{16}	2^{32}	$754b$ $8fa9_{16}$	0000 0000_{16}	-	3	2^{32}	$f8eb$ 8314 $dd04$ $6f67_{16}$	0000 0000 0000 0000_{16}	-	0	2^{32}	$cb8b$ $21b1$ 3524 9576 $767f$ $65f9$ $ec96$ $94f7_{16}$	0000 0000 0000 0000 0000 0000 0000 0000_{16}	-	0
			…	-				…	-				-	-	
			$754b$ $b8dd_{16}$	9502 $39a1_{16}$											
			$f5ef$ $8fa9_{16}$	9502 $39a1_{16}$											
			$f5ef$ $b8dd_{16}$	9502 $39a1_{16}$											
			…	-											
			$ffff_{16}$	-				$ffff$ $ffff$ $ffff$ $ffff_{16}$	-				$ffff$ $ffff$ $ffff$ $ffff$ $ffff$ $ffff$ $ffff$ $ffff_{16}$	-	

Tabel 1 Hasil Pengujian Fixed Point Attack pada Algoritme PRESENT

Kunci	PRESENT-16				PRESENT-32				PRESENT-64			
	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah
$0000\ 0000\ 0000\ 0000\ 0000_{16}$	2^{16}	0000_{16}	-	3	2^{32}	00000000_{16}	-	4	2^{32}	$00000000\ 00000000_{16}$	-	0
		0001_{16}	-			…	…			$00000000\ 00000001_{16}$	-	
		…	…			$0d2682b9_{16}$	$0d2682b9_{16}$			$00000000\ 00000002_{16}$	-	
		$0cec_{16}$	$0cec_{16}$			$dc70b70e_{16}$	$dc70b70e_{16}$			…	…	
		5523_{16}	5523_{16}			$e0c2e34c_{16}$	$e0c2e34c_{16}$			…	…	
		$cef5_{16}$	$cef5_{16}$			$e373c78a_{16}$	$e373c78a_{16}$			…	…	
		…	…			…	…			$00000000\ fffffffe_{16}$	-	
		$ffff_{16}$	-			$ffffffff_{16}$	-			$00000000\ ffffffff_{16}$	-	

Pada pengujian *second preimage attack* terhadap DOPE – [4], nilai *output* yang ditunjukkan sebagai hasil pengujian setelah tahapan permutasi pada penggunaan *chosen message* dan ketiga *input* yang menghasilkan nilai *hash* yang saling berkolisi menunjukkan hasil yang sama dengan posisi terbentuknya pesan *fixed point* pada [21]. Ketiga kolisi tersebut juga diketahui menghasilkan nilai yang sama setelah dilakukan operasi XOR antara hasil fungsi E dengan nilai *input*. Hal tersebut terjadi karena adanya dua pasang *input* berbeda yang menghasilkan operasi XOR yang sama seperti berikut.

$$56f3_{16} \oplus 754b_{16} = d657_{16} \oplus f5ef_{16} = 23b8_{16}$$
$$bf6b_{16} \oplus 8fa9_{16} = 881f_{16} \oplus b8dd_{16} = 30c2_{16}$$

Nilai tersebut selanjutnya akan menghasilkan nilai yang sama pula karena melalui tahapan transformasi linier beserta proses swap block. Selanjutnya pada kedua variasi lain algoritme DOPE, yaitu variasi dengan penggunaan basis fungsi enkripsi PRESENT berukuran 32-bit dan 64-bit diketahui tidak ditemukan adanya kolisi pada kedua metode pengujian yang diterapkan.

Tabel 2 Hasil Pengujian Fixed Point Attack pada Algoritme DOPE

DOPE – [4]					DOPE – [8]					DOPE – [16]				
IV	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah	IV	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah	IV	Ukuran Sampel Pengujian	*Input* pesan	*Fixed point*	Jumlah
$0cec_{16}$	2^{16}	$0000\ 0000_{16}$	-	0	$0d26\ 82b9_{16}$	2^{32}	$0000\ 0000\ 0000\ 0000_{16}$	-	0	$0000\ 0000\ 0000\ 0000_{16}$	2^{32}	$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{16}$	-	0
		…	…				…	…				…	…	
		$0000\ ffff_{16}$	-				$0000\ 0000\ ffff\ ffff_{16}$	-				$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ ffff\ ffff_{16}$	-	
5523_{16}	2^{16}	$0000\ 0000_{16}$	-	0	$dc70\ b70e_{16}$	2^{32}	$0000\ 0000\ 0000\ 0000_{16}$	-	0					
		…	…				…	…						

		0000 $ffff_{16}$	-				0000 0000 $ffff$ $ffff_{16}$	-		
$cef5_{16}$	2^{16}	0000 0000_{16}	-	0	$e0c2$ $e34c_{16}$	2^{32}	0000 0000 0000 0000_{16}	-	0	
		…	…				…	…		
		0000 $ffff_{16}$	-				0000 0000 $ffff$ $ffff_{16}$	-		
					$e373$ $c78a_{16}$	2^{32}	0000 0000 0000 0000_{16}	-	0	
							…	…		
							0000 0000 $ffff$ $ffff_{16}$	-		
Total				0	Total				0	Total 0

Tabel 6.Rekapitulasi Hasil Pengujian Algoritme DOPE

No	Varian Algoritme	Panjang pesan	Jumlah kolisi	
			Fixed point attack	*Second preimage attack*
1.	DOPE – [4]	32-bit	-	3
2.	DOPE – [8]	64-bit	-	-
3.	DOPE – [16]	128-bit	-	-

Tabel 6 menunjukkan keseluruhan hasil pengujian tanpa adanya iterasi pada seluruh variasi algoritme DOPE menggunakan metode *fixed point attack* dan *second preimage attack*. Berdasarkan pengujian menggunakan metode *fixed point attack* dan *second preimage attack* terhadap algoritme DOPE diketahui tidak ditemukan adanya kolisi. Tiga kolisi ditemukan pada implementasi *second preimage attack* terhadap variasi algoritme miniatur DOPE dengan basis fungsi enkripsi menggunakan algoritme PRESENT berukuran 16-bit. Ketiga kolisi terbentuk setelah proses operasi XOR antara hasil fungsi enkripsi E dengan nilai *input.* Implementasi s*econd preimage attack* pada dua iterasi DOPE – [4] dan *fixed point attack* terhadap satu iterasi DOPE – [4] terbukti tidak menghasilkan kolisi.

5. SIMPULAN

Berdasarkan penelitian yang dilakukan, diperoleh simpulan sebagai berikut:

a. Algoritme DOPE sebagai modifikasi dari skema MDC-2 berbasis PRESENT dapat memenuhi sifat *one-wayness* pada *hash function* dengan menunjukkan bahwa sulit untuk melakukan upaya *reverse* jika diketahui suatu nilai *hash* untuk mengetahui nilai *input* pesan. Jumlah kemungkinan usaha yang diperlukan untuk menemukan nilai *input* pada DOPE berbasis *block cipher* berukuran n-bit mencapai kemungkinan sejumlah $2 \times 2^n + 4n$.
b. Telah dilakukan pengujian menggunakan metode *fixed point attack* dan *second preimage attack* terhadap algoritme DOPE. Hasil pengujian menunjukkan bahwa kolisi hanya ditemukan pada metode pengujian *second preimage* attack berupa pencarian sifat *second preimage* pada variasi algoritme DOPE dengan basis fungsi enkripsi PRESENT berukuran 16-bit. Probabilitas ditemukannya kolisi pada variasi tersebut adalah sebesar $6.984919309616089 \times 10^{-10}$.

Berdasarkan hasil penelitian terdapat beberapa saran yang dapat diberikan untuk pengembangan penelitian selanjutnya sebagai berikut.

a. Melakukan pengujian dengan ukuran sampel yang lebih besar pada algoritme DOPE berbasis algoritme PRESENT berukuran 128-bit.
b. Diperlukan adanya pengujian terhadap kecepatan komputasi algoritme DOPE baik pada implementasi *hardware* maupun s*oftware*.
c. Penelitian lebih lanjut mengenai kemungkinan adanya kerawanan DOPE yaitu pada tahapan XOR antara hasil fungsi *E* dengan nilai *input*.

REFERENSI

[1] A. J. Menezes, P. C. van Oorschot, dan S. A. Vanstone, *Handbook Of Applied Cryptography*. United State: CRC Press, Inc.Subs. of Times Mirror 2000 Corporate Blvd. NW Boca Raton, FL, 1996.

[2] B. T. Hammad, N. Jamil, M. E. Rusli and M. R. Z`aba, "A survey of Lightweight Cryptographic Hash," *International Journal of Scientific & Engineering Research*, pp. 806 - 814, 2017

[3] W. Stallings, *Cryptography and network security : principles and practice*. Boston: Pearson Education. 2017.

[4] F. Mendel, “Analysis of Cryptographic Hash Functions,” Graz University of Technology, 2010.

[5] L. Pin, W. Wen-Ling and W. Chuan-Kun, "Hash Functions Based on Block Ciphers," *Journal of Software*, pp. 682 - 691, 2022.

[6] B. Preneel, “Cryptographic Primitives for Information Authentication-State of the Art,” 1998. doi: https://doi.org/10.1007/3-540-49248-8_3.

[7] C. Paar dan J. Pelzl, *Understanding Cryptography*. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-04101-3.

[8] L. R. Knudsen, • Matthew, dan J. B. Robshaw, *The Block Cipher Companion*. Springer Berlin, Heidelberg, 2011. doi: https://doi.org/10.1007/978-3-642-17342-4.

[9] T. S. Steffen, “General rights Cryptographic Hash Functions,” dalam Project: Ph.D.thesis Techinical University of Denmark. 2009.

[10] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, dan Y. Seurin, “Hash Functions and RFID Tags: Mind the Gap,” dalam *Cryptographic Hardware and Embedded Systems – CHES 2008*, 2008, hlm. 283–29. doi: https://doi.org/10.1007/978-3-540-85053-3_18.

[11] S. Hirose dan H. Kuwakado, “LNCS 8782 - A Block-Cipher-Based Hash Function Using an MMO-Type Double-Block Compression Function,” dalam *Provable Security. ProvSec 201471*, 2014, hlm. 71–86. doi: https://doi.org/10.1007/978-3-319-12475-9_6.

[12] J. W. Bos, O. Özen, dan M. Stam, “Efficient Hashing using the AES Instruction Set,” dalam *Cryptographic Hardware and Embedded Systems – CHES 2011*, 2011, hlm. 507–522. doi: https://doi.org/10.1007/978-3-642-23951-9_33.

[13] A. Bogdanov *dkk.*, “LNCS 4727 - PRESENT: An Ultra-Lightweight Block Cipher,” dalam *Cryptographic Hardware and Embedded Systems - CHES 2007. CHES 2007*, 2007, hlm. 450–466. doi: https://doi.org/10.1007/978-3-540-74735-2_31.

[14] A. Lutfiani dan B. H. Susanti, “DOPE: MDC-2 Scheme Using PRESENT,” dalam *Proceedings of the International Conference on Mathematics, Geometry, Statistics, and Computation (IC-MaGeStiC 2021)*, 2022, hlm. 215–221. doi: https://dx.doi.org/10.2991/acsr.k.220202.040.

[15] D. Engels, X. Fan, G. Gong, H. Hu, dan E. M. Smith, “Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices,” dalam *FC 2010: Financial Cryptography and Data Security*, 2010, hlm. 3–18. doi: https://doi.org/10.1007/978-3-642-14992-4_2.

[16] S. Hirose, “How to Construct Double-Block-Length Hash Functions,” dalam *Second Cryptographic Hash Workshop*, Santa Barbara. 2006.

[17] Y. S. S. Risqi, S. Yohanes, dan S. Windarta, “Fixed Point Attack in PGV-5 Scheme Using SIMON Algorithm,” dalam *Procedia Computer Science*, 2015, vol. 72, hlm. 292–299. doi: 10.1016/j.procs.2015.12.143.

[18] N. S. S. S. Winarno A, “Fixed Point Attack pada Simplified IDEA dengan Skema Davies Meyer,” *Seminar Nasional Teknologi Informasi dan Multimedia*, 2018.

[19] J. Kelsey dan B. Schneier, “Second Preimages on n-bit Hash Functions for Much Less than 2 n Work,” dalam *Advances in Cryptology – EUROCRYPT 2005*, 2005, hlm. 474–490. doi: https://doi.org/10.1007/11426639_28.

[20] E. Fleischmann, M. Gorski, dan S. Lucks, “Security of Cyclic Double Block Length Hash Functions including Abreast-DM,” dalam *IMACC 2009: Cryptography and Coding*, 2009, hlm. 153–175. doi: https://doi.org/10.1007/978-3-642-10868-6_10.

[21] T. A. Darumaya, “Forgery Attack pada skema LightMAC berbasis Algoritme SIMECK 32/64 dan Small PRESENT [n] serta Fixed Point pada Underlying Block Cipher,” *Sekolah Tinggi Sandi Negara (unpublished)*. 2018.